2D Generalization of Fractional Cascading on Axis-aligned Planar Subdivisions

Peyman Afshani[*] and Pingan Cheng[*]
Department of Computer Science
Aarhus University
Aarhus, Denmark
Email: {peyman, pingancheng}@cs.au.dk

Abstract—**Fractional cascading is one of the influential and important techniques in data structures, as it provides a general framework for solving a common important problem: the iterative search problem. In the problem, the input is a graph G with constant degree. Also as input, we are given a set of values for every vertex of G. The goal is to preprocess G such that when we are given a query value q, and a connected subgraph π of G, we can find the predecessor of q in all the sets associated with the vertices of π. The fundamental result of fractional cascading, by Chazelle and Guibas, is that there exists a data structure that uses linear space and it can answer queries in $O(\log n + |\pi|)$ time, at essentially constant time per predecessor [16]. While this technique has received plenty of attention in the past decades, an almost quadratic space lower bound for "two-dimensional fractional cascading" by Chazelle and Liu in STOC 2001 [18] has convinced the researchers that fractional cascading is fundamentally a one-dimensional technique.**

In two-dimensional fractional cascading, the input includes a planar subdivision for every vertex of G and the query is a point q and a subgraph π and the goal is to locate the cell containing q in all the subdivisions associated with the vertices of π. In this paper, we show that it is actually possible to circumvent the lower bound of Chazelle and Liu for axis-aligned planar subdivisions. We present a number of upper and lower bounds which reveal that in two-dimensions, the problem has a much richer structure. When G is a tree and π is a path, then queries can be answered in $O(\log n + |\pi| + \min\{|\pi|\sqrt{\log n}, \alpha(n)\sqrt{|\pi|}\log n\})$ time using linear space where α is an inverse Ackermann function; surprisingly, we show both branches of this bound are tight, up to the inverse Ackermann factor. When G is a general graph or when π is a general subgraph, then the query bound becomes $O(\log n + |\pi|\sqrt{\log n})$ and this bound is once again tight in both cases.

Keywords-**Fractional Cascading; Computational Geometry; Data Structures and Algorithms**

I. INTRODUCTION

Fractional cascading [16] is one of the widely used tools in data structures as it provides a general framework for solving a common important problem: the iterative search problem, i.e., the problem of finding the predecessor of a single value q in multiple data sets. In the problem, we are to preprocess a degree-bounded "catalog" graph G where each vertex represents an input set of values from a totally ordered universe U; the input sets of different vertices of G are completely unrelated. Then, at the query time, given a value $q \in U$ and a connected subgraph π of G, the goal is to find the predecessor of q in the sets that correspond to the vertices of π. The fundamental theorem of fractional cascading is that one can build a data structure of linear size such that the queries can be answered in $O(\log n + |\pi|)$ time, essentially giving us constant search time per predecessor after investing an initial $O(\log n)$ search time [16]. Many problems benefit from this technique [17] since they need to solve the iterative search problem as a base problem.

Given its importance, it is not surprising that many have attempted to generalize this technique: The first obvious direction is to consider the dynamic version of the problem by allowing insertions or deletions into the sets of the vertices of G. In fact, Chazelle and Guibas themselves consider this [16] and they show that with $O(\log n)$ amortized time per update, one can obtain $O(\log n + |\pi| \log \log n)$ query time. Later, Mehlhorn and Näher improve the update time to $O(\log \log n)$ amortized time [26] and then Dietz and Raman [22] remove the amortization. There is also some attention given to optimize the dependency of the query time on the maximum degree of graph G [23].

The next obvious generalization is to consider the higher dimensional versions of the problem. Here, each vertex of G is associated with an input subdivision and the goal is to locate a given query point q on every subdivision associated with the vertices of π. Unfortunately, here we run into an immediate roadblock already in two dimensions: After listing a number of potential applications of two-dimensional fractional cascading, Chazelle and Liu [18] "dash all such hopes" by showing an $\tilde{\Omega}(n^2)$ [1] space lower bound in the pointer-machine model for any data structure that can answer queries in $O(\log^{O(1)} n + |\pi|)$ time. Note that this lower bound can be generalized to also give a $\Omega(n^{2-\varepsilon})$ space lower bound for data structures with $O(\log^{O(1)} n) + o(|\pi| \log n)$ query time. As far as we can tell, progress in this direction was halted due to this negative result since the trivial solution already gives the $O(|\pi| \log n)$ query time, by just building individual point location data structures for each subdivision.

We observe that the lower bound of Chazelle and Liu does not apply to orthogonal subdivisions, a very important spe-

---

[1]The $\tilde{\Omega}$ notation hides polylogarithmic factors.

cial case of planar point location problem. Many geometric problems need to solve this base problem, e.g., 4D orthogonal dominance range reporting [3], [4], 3D point location in orthogonal subdivisions [27], some 3D vertical ray-shooting problems [21]. In geographic information systems, it is very common to overlay planar subdivisions describing different features of a region to generate a complete map. Performing point location queries on such maps corresponds to iterative point locations on a series of subdivisions.

Motivated by this observation, we systematically study the generalization of fractional cascading to two dimensions, when restricted to orthogonal subdivisions. We obtain a number of interesting results, including both upper and lower bounds which show most of our results are tight except for the general path queries of trees where the bound is tight up to a tiny inverse Ackermann factor [19].

The problem definition: The formal definition of the problem is as follows. The input is a degree-bounded connected graph $G = (V, E)$ where each vertex $v \in V$ is associated with an axis-aligned planar subdivision. Let n be the total number of vertices, edges, and faces in the subdivisions, which we call the *graph subdivision complexity*. We would like to build a data structure such that given a query (q, π), where q is a query point and π is a connected subgraph of G, we can locate q in all the subdivisions induced by vertices of π efficiently. We call this problem 2D Orthogonal Fractional Cascading (2D OFC).

A. Related Work

While the negative result of Chazelle and Liu [18] stops any progress on the general problem of 2D fractional cascading, there have been other results that can be seen as special cases of two-dimensional fractional cascading. For example, Chazelle et al. [14] improved the result of ray shooting in a simple polygon by a $\log n$ factor. In a "geodesically triangulated" subdivision of n vertices, they showed it is possible to locate all the triangles crossed by a ray in $O(\log n)$ time instead of $O(\log^2 n)$, which resembles 2D fractional cascading. However, their solution relies heavily on the characteristic of geodesic triangulation and cannot be generalized to other problems. Chazelle's data structure for the rectangle stabbing problem [12] can also be viewed as a restricted form of 2D fractional cascading where $\pi = G$.

In recent years, interestingly, a technique similar to 2D fractional cascading has been used to improve many classical computational geometry data structures. While working on the 4D dominance range reporting problem, Afshani, Arge, and Larsen [3] are implicitly performing iterative point location queries along a path of a balanced binary tree on somewhat specialized subdivsions in $O(\log^{3/2} n)$ total time. Later Afshani, Chan, and Tsakalidis [4] studied an offline variant of this problem, and they presented a linear sized data structure that achieves optimal query time. The same idea is used to improve the result of 3D point location in orthogonal

subdivisions. In that probelm, Rahul [27] obtained another data structure with $O(\log^{3/2} n)$ query time.

Another related problem is the "unrestricted" version of fractional cascading where essentially π can be an arbitrary subgraph of G, instead of a connected subgraph. In one variant, we are given a set L of categories and a set S of n points in d dimensional space where each point belongs to one of the categories. The query is given by a d-dimensional rectangle r and a subset $Q \subset L$ of the categories. We are asked to report the points in S contained in r and belonging to the categories in Q. In 1D, Chazelle and Guibas [17] provided a $O(|Q| \log \frac{|L|}{|Q|} + \log n + k)$ query time and linear size data structure, where k is the output size, together with a restricted lower bound. Afshani, Sheng, Tao, and Wilkinson [5] strenghtened the lower bound and presented several data structures for three-sided queries in two-dimensions. Their data structures match the lower bound within an inverse Ackermann factor for the general case.

B. Our Results

We study 2D OFC in a pointer machine model of computation. Some of our bounds involve inverse Ackermann functions. The particular definition that we use is as follows: We define $\alpha_2(n) = \log n$ and then we define $\alpha_i(n) = \alpha_{i-1}^*(n)$, meaning, it's the number of times we need to apply the $\alpha_{i-1}(\cdot)$ function to n until we reach a fixed constant. $\alpha(n)$ corresponds to the value of i such that $\alpha_i(n)$ is at most a fixed constant. Our results are summarized in Table I.

Our results show some very interesting behavior. First, by looking at the last two rows of Table I, we see that we can always do better than the naïve solution by a $\sqrt{\log n}$ factor. Furthermore, this is tight. We show matching query lower bounds both when G is an arbitrary graph but π is restricted to be a path and also when G is a tree but π is allowed to be any subtree of G. Second, when G is a tree and π is a path we get some variation depending on the query path length. When π is of length at most $\frac{\log n}{2}$, then we can answer queries in $O(|\pi|\sqrt{\log n})$ time, but when π is longer than $\frac{\log n}{2}$, we obtain the query bound of $O(\sqrt{|\pi|} \log n)$ (ignoring some inverse Ackermann factors). Furthermore, we give two lower bounds that show both of these branches are tight! When π is very long, longer than $\frac{\log^2 n}{2}$, the query bound becomes $O(|\pi|)$ which is also clearly optimal.

II. PRELIMINARIES

In this section, we introduce some geometric preliminaries and present the tools we will use to build the data structures and to prove the lower bounds.

A. Geometric Preliminaries

We first review some concepts related to cuttings.

Definition II.1. *Given a set H of n hyperplanes in the plane, a $(1/r)$-cutting, $1 \le r \le n$, is a set of (possibly open) disjoint simplices that together cover the entire plane such*

Table I: Our Results

Graph	Query	Space	Query Time	Tight?
Tree	Path	$O(n\alpha_c(n))$	$O(\min\{|\pi|\sqrt{\log n}, c\sqrt{|\pi|}\log n\} + \log n + |\pi|)$	Up to $\alpha_c(n)$ factor
Tree	Path	$O(n)$	$O(\min\{|\pi|\sqrt{\log n}, \alpha(n)\sqrt{|\pi|}\log n\} + \log n + |\pi|)$	Up to $\alpha(n)$ factor
Tree	Subtree	$O(n)$	$O(\log n + |\pi|\sqrt{\log n})$	yes
Graph	Path / Subgraph	$O(n)$	$O(\log n + |\pi|\sqrt{\log n})$	yes

that each simplex intersects $O(n/r)$ hyperplanes of H. For each simplex in the cutting, the set of all hyperplanes of H intersecting it is called the conflict list of that simplex.

$(1/r)$-cuttings are important in computational geometry as they enable us to apply the divide-and-conquer paradigm in higher dimensions. The following theorem by Chazelle [13], after a series of work in the computational geometry community [6], [7], [15], [24], [25], shows the existence of $(1/r)$-cuttings of small size and an efficient deterministic algorithm computing $(1/r)$-cuttings.

Theorem II.1 (Chazelle [13]). *Given a set H of n hyperplanes in the plane, there exists a $(1/r)$-cutting, $1 \le r \le n$, of size $O(r^2)$, which is optimal. We can find the cutting and the conflict lists in $O(nr)$ time.*

In this paper, we will use intersection sensitive $(1/r)$-cuttings which is a generalization of $(1/r)$-cuttings. The following theorem is given by de Berg and Schwarzkopf [9].

Theorem II.2 (de Berg and Schwarzkopf [9]). *Given a set H of n line segments in the plane with A intersections, we can construct a $(1/r)$-cutting, $1 \le r \le n$, of size $O(r + Ar^2/n^2)$. We can find the cutting and the conflict lists in time $O(n \log r + Ar/n)$ by a randomized algorithm.*

Note that by the construction of generalized cuttings, see [9] for detail, the following corollary follows directly from Theorem II.2,

Corollary II.1. *Given an axis-aligned planar subdivision of complexity n, we can construct a $(1/r)$-cutting, $1 \le r \le n$, of size $O(r)$. Furthermore, each cell of the cutting is an axis-aligned rectangle and the size of the conflict list of every cell is bounded by $O(n/r)$. We can find the cutting and the conflict lists in time $O(n \log n)$ by a randomized algorithm.*

B. Rectangle Stabbing

In d-dimensional rectangle stabbing problem, we are given a set of n d-dimensional axis-parallel rectangles, our task is to build a data structure such that given a query point q, we can report the rectangles containing the query point efficiently. As noted earlier, Chazelle [12] provides an optimal solution in two-dimensions, a linear-sized data structure that can answer queries in $O(\log n + t)$ time where t is the output size. The following lemma by Afshani, Arge, and Larsen [3] establishes an upper bound of this problem

and it is obtained by a basic application of range trees [8] with large fan-out and Chazelle's data structure.

Lemma II.1 (Afshani, Arge, and Larsen [3]). *We can answer d dimensional rectangle stabbing queries in time $O(\log n \cdot (\log n / \log H)^{d-2} + t)$ using space $O(nH \log^{d-2} n)$, where n is the number of rectangles, t is the output size, and $H \ge 2$ is any parameter.*

C. A Pointer Machine Lower Bound Framework

We will use the pointer machine lower bound framework of Afshani [1]. The framework deals with an abstract "geometric stabbing problem" which is defined by a set $\mathcal{R}$ of "ranges" and a set U of queries. An instance of the geometric stabbing problem is given by a set $R \subset \mathcal{R}$ of n "ranges" and the goal is to preprocess R to answer queries q. Given R, an element $q \in U$ (implicitly) defines a subset $R_q \subset R$ and the data structure is to output the elements of R_q. However, the data structure is restricted to operate in the (strengthened) pointer machine model of computation where the memory is a directed graph M consisting of "cells" where each cell can store an element of R as well as two pointers to other memory cells. At the query time, the algorithm must find a connected subgraph M_q of M where each element of R_q is stored in at least one memory cell of M_q. The size of M is a lower bound on the space complexity of the data structure and the size of R_q is a lower bound on the query time. However, the lower bound model allows for unlimited computation and allows the data structure to have complete information about the problem instance; the only bottleneck is being able to navigate to the cells storing the output elements. In addition, the framework assumes that we have a measure μ such that $\mu(U) = 1$. We need a slightly more precise version of the lower bound framework where the dependency on a certain "constant" is made explicit.

Theorem II.3. *Assume, we have an algorithm that given any input instance $R \subset \mathcal{R}$ of n ranges, it can store R in a data structure of size $S(n)$ such that given any query $q \in U$, it can answer the query in $Q(n) + \gamma|R_q|$ time.*

Then, suppose we can construct an input set $R \subset \mathcal{R}$ of n ranges such that the following two conditions are satisfied: (i) every query point $q \in U$ is contained in exactly $|R_q| = t$ ranges and $\gamma t \ge Q(n)$; (ii) there exists a value v such that for any two ranges $r_1, r_2 \in R$, $\mu(\{q \in U | r_1, r_2 \in R_q\})$ is well-defined and is upper bounded by v. Then, we must have $S(n) = \Omega(tv^{-1}/2^{O(\gamma)}) = \Omega(Q(n)v^{-1}/2^{O(\gamma)})$.

For the proof of this theorem, we refer the readers to the full version of this paper. In our applications, μ will basically be the Lebesgue measure and U will be the unit cube.

III. Queries on Catalog Paths

In this section, we give a simple solution for when the catalog graph is a path. It will be used as a building block for later data structures.

Theorem III.1. *Consider a catalog path G, in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can construct a data structure using $O(n)$ space such that given any query (q, π), where q is a query point and π is a subpath, all regions containing q along π can be reported in time $O(\log n + |\pi|)$.*

Proof: We can convert each subdivision into a set of disjoint rectangles of total size $O(n)$ using trapezoidal decomposition [20]. Then, we partition G into $m = \lceil |G| / \log n \rceil$ paths, $G_1, \cdots, G_m$ where each path except potentially for G_m has size $\log n$ and G_m has size at most $\log n$.

Now we use an observation that was also made in previous papers [2], [3], [27]: when $H = G$, the two-dimensional fractional cascading can be reduced to rectangle stabbing. As a result, for each G_i, $1 \leq i \leq m$, we collect all the rectangles of its subdivisions and build a 2D rectangle stabbing data structure on them. By Lemma II.1 this requires $O(n)$ space. Now given a query subpath of length $|\pi|$, we use the rectangle stabbing data structures on the subdivisions of each G_i as long as $|G_i \cap \pi| > 0$. Since π is a path, for at most two indices i we will have $0 < |G_i \cap \pi| < \log n$ and for the rest $|G_i \cap \pi| = |G_i| = \log n$. This gives us $O(\log n + |\pi|)$ query time. ∎

IV. Path Queries on Catalog Trees

Now we consider answering path queries on catalog trees. We first show optimal data structures for trees of different heights. It turns out we need different data structures to achieve optimality when heights differ. We then present a data structure using $O(n\alpha_c(n))$ space that can answer path queries in $O(\log n + |\pi| + \min\{|\pi|\sqrt{\log n}, \sqrt{|\pi|}\log n\})$ time and a data structure using $O(n)$ space answering path queries in $O(\log n + |\pi| + \min\{|\pi|\sqrt{\log n}, \alpha(n)\sqrt{|\pi|}\log n\})$ time, where $c \geq 3$ is any constant and $\alpha_k(n)$ is the k-th function in the inverse Ackermann hierarchy [19] and $\alpha(n)$ is the inverse Ackermann function [19]. We also present lower bounds for our data structures. Without loss of generality, we assume the tree is a binary tree.

A. Trees of height $h \leq \frac{\log n}{2}$

1) The Upper Bound: For trees of this height, we present the following upper bound. The main idea is to use the sampling idea that is employed previously [3], [10], however, there are some main differences. Instead of random samples or shallow cuttings, we use intersection sensitive cuttings [9]

and more notably, the fractional cascading on an arbitrary tree cannot be reduced to a geometric problem such as 3D rectangle stabbing, so instead we do something else.

Lemma IV.1. *Consider a catalog tree of height $h \leq \frac{\log n}{2}$ in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can build a data structure using $O(n)$ space such that given any query (q, π), where q is a query point and π is a path, all regions containing q along π can be reported in time $O(\log n + |\pi|\sqrt{\log n})$.*

Proof: Let r be a parameter to be determined later. Consider a planar subdivision A_i and let n_i be the number of rectangles in A_i. We create an intersection sensitive (r^2/n_i)-cutting C_i on A_i. By Corollary II.1, C_i contains $O(n_i/r^2)$ cells and each cell of C_i is an axis-aligned rectangle. Furthermore, the conflict list size of each rectangle is $O(r^2)$. For each cell in C_i, we build an optimal point location data structure on its conflict list. The total space usage is linear, since total size of the conflict lists is linear.

Then we consider every path of length at most $\log r$ in the catalog graph, and we call them subpaths. For every subpath, we collect all the cells of the cuttings belonging to the vertices of the subpath and build a 2D rectangle stabbing data structure on them. Since the degree of any vertex is bounded by 3, each vertex is contained in at most

$$\sum_{j=0}^{\log r} \sum_{i=0}^{j} 3^i \cdot 3^{j-i} = \Theta(r^{\log 3} \log r)$$

subpaths. Then the total space usage of the 2D rectangle stabbing data structures is bounded by $O(n \log r / r^{2-\log 3}) = O(n)$. Given any query path π, it can be covered by $|\pi|/\log r$ subpaths. For each subpath, we can find all the cells of the cuttings containing the query point in $O(\log n)$ time and then do an additional point location on its conflict list, for a total of $O(\log n + (\log r)^2)$ query time per subpath. Thus, the query time of this data structure is bounded by

$$\frac{|\pi|}{\log r}(\log n + \log r \cdot \log r).$$

We pick $r = 2^{\sqrt{\log n}}$, then we obtain the desired $O(\log n + |\pi|\sqrt{\log n})$ query time. ∎

2) The Lower Bound: We now present a matching lower bound. We show the following:

Lemma IV.2. *Assume, given any catalog tree of height $\sqrt{\log n} \leq h \leq \frac{\log n}{2}$ in which each vertex is associated with a planar subdivision with n being the total complexity of the subdivisions, we can build a data structure that satisfies the following: it uses at most $n2^{\varepsilon\sqrt{\log n}}$ space, for a small enough constant ε, and it can answer 2D OFC queries (q, π). Then, its query time must be $\Omega(|\pi|\sqrt{\log n})$.*
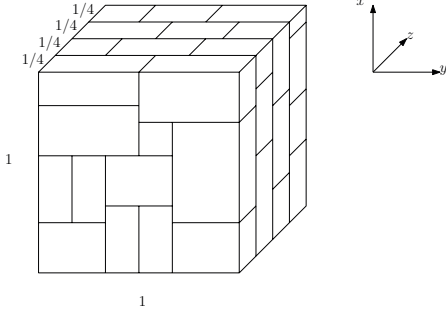
Figure 1: An example of rectangles in set 2

Proof: We will use the following idea: We consider a special 3D rectangle stabbing problem and show a lower bound using Theorem II.3. We will use the 3D Lebesgue measure, denoted by $V(\cdot)$. Then we show a reduction from this problem to a 2D OFC problem on trees to obtain the desired lower bound.

We consider the following instance of 3D rectangle stabbing problem. The input n rectangles are partitioned into h sets of size n/h each. The rectangles of each set are pairwise disjoint and they tile the unit cube in 3D. The depth (i.e., the length of the side parallel to the z-axis) of rectangles in set $i = 0, \cdots, h-1$ is $1/2^i$. In fact, the rectangles of each set can be partitioned into 2^i subsets where the projection of the rectangles in the j-th subset onto the z-axis is the interval $[\frac{j}{2^i}, \frac{j+1}{2^i}]$. See Figure 1 for an example.

We first show the reduction: assume, we are given an instance of special 3D rectangle stabbing problem. We build a balanced binary tree of height h on the z-axis as the catalog graph. Note that the number of vertices at layer i of the tree is the same as the number of subsets in set i. We project the rectangles in each subset to the xy-plane and obtain a 2D axis-aligned planar subdivision. We attach each of the subdivisions to the corresponding vertices. Consider a 2D OFC query, (q, π) in which π is a path that connects the root to a lead. We lift q to a point q' in 3D appropriately: W.l.o.g, assume the leaf is the j-th leaf. To obtain q', we assign the z coordinate $\frac{j+0.5}{2^{h-1}}$ to q. By our construct, the z-axis projection of any rectangle in the node from the root to the leaf contains the z coordinate of q'. This construction ensures that finding the rectangles that contain q' is equivalent to performing 2D OFC query (q, π).

Now we describe a hard instance of the special rectangle stabbing problem to establish a lower bound. It will have rectangles of h different shapes. For each shape, we tile (disjointly cover) the unit cube using isometric copies of the shape to obtain a set of rectangles. We collect every r different shapes into a class and obtain h/r classes, where r is a parameter to be determined later. We say that the i-th rectangle in a class has group number i, $0 \leq i \leq r-1$. Now we specify the dimensions (i.e., side lengths) of the rectangles. For a rectangle in class $i = 0, \cdots, h/r-1$, with

group number $j = 0, \cdots, r-1$, its dimensions are

$$[\frac{1}{K^j} \times K^j \cdot 2^{ir+j} \cdot V \times \frac{1}{2^{ir+j}}],$$

where K, V are parameters to be determined later and the $[W \times H \times D]$ notation denotes an axis-aligned rectangle with width W, height H, and depth D. Observe that every rectangle has volume V and thus we need $1/V$ copies to tile the unit cube. By setting $V = h/n$, the total number of rectangles we generate is n. Also note that all the rectangles in the same group are pairwise disjoint and they together cover the whole unit cube. This implies for any query point q in the unit cube, it is contained in exactly $h = |\pi|$ rectangles.

Now we analyze the intersection of any two rectangles. First, observe that given two axis-aligned rectangles with dimensions $[W_1 \times H_1 \times D_1]$ and $[W_2 \times H_2 \times D_2]$, their intersection is an axis-aligned rectangle with dimensions at most $[\min\{W_1, W_2\} \times \min\{H_1, H_2\} \times \min\{D_1, D_2\}]$. Second, by our construction, the rectangles that have identical width, depth, and height are disjoint. As a result, either the width of the two rectangles will differ by a factor K or their depth will differ by a factor 2^r. This means that, the maximum intersection volume of any two rectangles R_1, R_2 in class i_1, i_2, group j_1, j_2 can be achieved only in one of the following two cases:

$$V(R_1 \cap R_2) = \begin{cases} \frac{V}{2K} & i_1 = i_2 \text{ and } j_1 = j_2 + 1, \\ \frac{V}{2^r} & i_1 = i_2 + 1 \text{ and } j_1 = j_2. \end{cases}$$

We set $K = 2^r$, then the intersection volume of any two rectangles is bounded by $v = V/2^r$. However, for the construction to be well-defined, the side length of the rectangles cannot exceed 1 as otherwise, they do not fit in the unit cube. The largest height of the rectangles is obtained for $j = r-1$ and $i = \frac{h}{r} - 1$. Thus, we must have,

$$K^{r-1} 2^{r(\frac{h}{r}-1)+r-1} V \leq 1.$$

Plugging $V = h/n$ and $K = 2^r$ we get that we must have

$$2^{r^2-r} 2^{h-1} h < n. \tag{1}$$

Since by our assumptions $h \leq \frac{\log n}{2}$, it follows that by setting $r = \frac{\sqrt{\log n}}{4}$, the inequality (1) holds.

If $\gamma h \geq Q(n)$ holds, then we satisfy the first condition of Theorem II.3 and thus we obtain the space lower bound of

$$S(n) = \Omega\left(\frac{Q(n)v^{-1}}{2^{O(\gamma)}}\right) = \Omega\left(\frac{n2^r}{\log n 2^{O(\gamma)}}\right). \tag{2}$$

If we set $\gamma = \delta\sqrt{\log n}$ for a sufficiently small $\delta > 0$, it follows that the data structure must use $\Omega(n2^{\Omega(\sqrt{\log n})})$ space. However, by the statement of our lemma, we are not considering such data structures. As a result, when $\gamma = \delta\sqrt{\log n}$, the query time must be large enough that the first condition of the framework does not hold, meaning, we must have $Q(n) \geq \gamma h = \delta\sqrt{\log n}h = \Omega(|\pi|\sqrt{\log n})$.

∎

B. Trees of height $\frac{\log n}{2} < h \leq \frac{\log^2 n}{2}$

1) The Upper Bound: We start with the following lemma which gives us a data structure that can only answer query paths that start from the root and finish at a leaf. The main idea here is used previously in the context of four-dimensional dominance queries [3], [10] and it uses the observation that such "root to leaf" queries can be turned into a geometric problem, the 3D rectangle stabbing problem.

Lemma IV.3. *Consider a balanced catalog tree of height h, $\frac{\log n}{2} < h \leq \frac{\log^2 n}{2}$, in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can build a data structure using $O(n)$ space such that given any query (q, π), where q is a query point and π is a path starting from the root to a leaf, all regions containing q along π can be reported in time $O(\sqrt{|\pi|} \log n)$.*

Proof: Let r be a parameter to be determined later. For each subdivision A_i, we create an intersection sensitive (r/n_i)-cutting C_i on A_i. By the same argument as Lemma IV.1, all the cells in the cuttings are axis-aligned rectangles satisfying (i) the conflict set size of any cell in C_i is bounded by $O(r)$ and (ii) the total number of cells in C_i is $O(n_i/r)$.

Now we lift each cell in the cuttings to 3D rectangles and collect all the 3D rectangles to construct a 3D rectangle stabbing data structure for it. This is done as follows. We assign a z range for each vertex in the catalog tree; Let m be the number of leaves. Order the leaves of the catalog tree from left to right and for the i-th leaf $l_i, i \in \{1, 2, \cdots, m\}$, we assign the range $[i-1, i)$ as its z range. For any internal vertex, its z range is the union of the z ranges of its children. Then, we lift the 2D rectangles induced by the subdivision of a vertex to a 3D rectangle using the z range (i.e., by forming the Cartesian product of the rectangle and the z range). We store the 3D rectangles in a rectangle stabbing data structure. Given a query point $q = (x_q, y_q)$ and a query path π, we first lift q to be (x_q, y_q, z_q), where z_q is any z value in the z range of the deepest vertex in π, and then query the 3D rectangle stabbing data structure.

For each cell in a cutting, we build an optimal point location data structure on its conflict set. These data structures take space $O(\sum_i n_i) = O(n)$ in total and each of them can answer a point location query in time $O(\log r)$.

To achieve space bound $O(n)$ for the 3D rectangle stabbing data structure, it suffices to choose $H = \frac{r}{\log(n/r)}$. We then balance the query time for 3D rectangle stabbing and 2D point locations to achieve the optimal query time

$$\log n \cdot \frac{\log n}{\log \frac{r}{\log(n/r)}} = h \cdot \log r.$$

We pick $r = 2^{\log n / \sqrt{h}}$ and the query time is bounded by $O(\sqrt{h} \log n) = O(\sqrt{|\pi|} \log n)$. ∎

The above data structure is not a true fractional cascading data structure because it can only support restricted queries. To be able to answer query paths of arbitrary lengths $> \frac{\log n}{2}$ and $\leq \frac{\log^2 n}{2}$, we need the following result.

Lemma IV.4. *Consider a catalog tree in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions and let h_1 and h_2, $h_1 < h_2$, be two fixed parameters. We can build a data structure using $O(n \log(h_2/h_1))$ space such that given any query (q, π), where q is a query point and π is a path whose length obeys $h_1 \leq |\pi| \leq h_2$, all regions containing q along π can be reported in time $O(\sqrt{|\pi|} \log n)$.*

Proof: First, observe that w.l.o.g, we can assume that the height of the catalog tree is at most h_2: we can partition the catalog tree into a forest by cutting off vertices whose depth is a multiple of h_2. Since the length of π is at most h_2, it follows that π can only contain vertices from at most two of the trees in the resulting forest, meaning, answering π can be reduced to answering at most two queries on trees of height at most h_2.

Thus, w.l.o.g, assume v is the root of the catalog tree of height h_2 and π is a path of length at least h_1 in this catalog tree. We build the following data structures. Let $v_1, \cdots, v_m$ be the vertices at height $h_2/2$. Let T_0 be the tree rooted at v and cut off at height $h/2$ with $v_1, \cdots, v_m$ being leafs and T_i be the tree rooted at v_i, $1 \leq i \leq m$. We build $m+1$ data structures of Lemma IV.3 on $T_0, \cdots, T_m$ and then we recurse on each of the $m+1$ trees. The recursion stops once we reach subproblems on trees of height at most h_1.

Since the data structure of Lemma IV.3 uses $O(n)$ space, at each recursive level, the total space usage of data structures we constructed is $O(n)$. Over the $O(\log(h_2/h_1))$ recursion levels, this sums up to $O(n \log(h_2/h_1))$ space.

Now we analyze the query time. Given a query (q, π), we may query several data structures that together cover the whole path of π. Let w be the highest vertex on π. We can decompose π into two disjoint parts π_1 and π_2, that start from w and end at vertices u_1 and u_2 respectively, with u_1 and u_2 being descendants of w. It thus suffices to only answer π_1, as the other path can be answered similarly. The first observation is that we can find a series of data structures that can be used to answer disjoint parts of π_1. The second observation is that we can afford to make the path a bit longer to truncate the recursion. We now describe the details.

Consider the trees $T_0, \cdots, T_m$ defined at the top level of the recursion. If π_1 is entirely contained in one of the trees, then we recurse on that tree. Otherwise, w is contained in T_0 and u_1 is contained in some subtree T_i. Now, π_1 can be further subdivided into two smaller "anchored" paths: one from w to v_i ("anchored" at w) and another from v_i to u_1 ("anchored" at v_i) and each smaller path can be answered recursively in the corresponding tree. Thus, it suffices to consider answering the query q along an anchored path.

Thus, consider the case of answering an anchored path π' in the data structure. To reduce the notation and clutter, assume π' is an anchored path, starting from the root of T and ending at a vertex u. Assume the vertices $v_1, \cdots, v_m$ and trees $T_0, \cdots, T_m$ are defined as above. First, consider the case when the height of T is at most h_1; in this case, we have built an instance of the data structure of Lemma IV.3 on T but not on the trees $T_0, \cdots, T_m$. In this case, we simply answer q on a root of leaf path in T that includes π', e.g., by a picking a leaf in the subtree of u. In this case, we will be performing a number of "useless" point location queries, in particular those on the descendants of u. However, as the height of T is at most h_1, it follows that the query bound stays asymptotically the same: $O(\sqrt{h_1}\log n)$. Furthermore, there is no recursion in this case and thus this cost is paid only once per anchored path. The second case is when the height of T is greater than h_1. In this case, if u lies in T_0 we simply recurse on T_0 but if u lies in a tree T_i, we first query the data structure of Lemma IV.3 using the path from the root of T until v_i, and then we recurse on T_i. As a result, answering the anchored path query reduces to answering at most one query on an instance of data structure Lemma IV.3 and another recursive "anchored" on a tree of half the height. Thus, the i-th instance of the data structure Lemma IV.3 that we query covers at most $1/2^i$ fraction of the anchored path. Thus, if k is the length of the anchored path, it follows that the total query time of all the data structures we query is bounded by

$$\sum_{i=1}^{\infty} \frac{\sqrt{k}\log n}{2^i} = O(\sqrt{k}\log n) = O(\sqrt{|\pi|}\log n).$$

We now reduce the space of the above lemma dramatically. We will repeatedly use a "bootstrapped" data structure. The following lemma establishes how we can bootstrap a base data structure to obtain a more efficient one.

Lemma IV.5. *Consider a catalog tree of height h, $\frac{\log n}{2} < h \leq \frac{\log^2 n}{2}$, in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. Assume, for any fixed value Δ, $\omega(1) \leq \Delta \leq \log n$, we can build a "base" data structure that can answer a 2D OFC query (q, π) in $Q_b(n) = O(\sqrt{|\pi|}\log n)$ time as long as π is path of length between $\frac{\log^2 n}{2\Delta}$ and $\frac{\log^2 n}{2}$. Furthermore, assume it uses $S_b(\Delta, n) = O(nf(\Delta))$ space, for some function f which is monotone increasing in Δ and for $\Delta = \omega(1)$ we have $f(\Delta) = \omega(1)$.*

Then, for any given fixed value Δ, $\omega(1) \leq \Delta \leq \log n$, we can build a "bootstrapped" data structure that can answer a 2D OFC query (q, π) in $Q_b(n) + O(\sqrt{|\pi|}\log n)$ time as long as π is path of length between $\frac{\log^2 n}{2\Delta}$ and $\frac{\log^2 n}{2}$. Furthermore, it uses $O(nf^*(\Delta))$ space, where $f^*(\cdot)$ is the iterative $f(\cdot)$ function which denotes how many times we need to apply $f(\cdot)$ function to Δ to reach a constant value.

Proof: We construct an intersection sensitive $(f(\Delta)/n_i)$-cutting C_i for each planar subdivision A_i attached to the tree. Call these the "first level" cuttings. Similar to the analysis in Lemma IV.1, we obtain $O(n_i/f(\Delta))$ cells, which are disjoint axis-aligned rectangles, for each C_i and $n' = O(n/f(\Delta))$ cells in total. Each cell in the cutting has a conflict list of size $O(f(\Delta))$ and on that we build a point location data structure. This takes $O(n)$ space in total. We store the cells of the cutting in an instance of the base data structure with parameter Δ. Call this data structure $\mathcal{A}_1$. The space usage of $\mathcal{A}_1$ is

$$S_b(\Delta, n') = O(n'f(\Delta)) = O(n).$$

Now we consider a query (q, π). Let $\delta_1 = \log(f(\Delta))$. Consider the case when $\frac{1}{2} \cdot \left(\frac{\log^2 n}{\Delta}\right) \leq |\pi| \leq \frac{1}{2} \cdot \left(\frac{\log n}{\delta_1}\right)^2$. In this case, as $\mathcal{A}_1$ is built with parameter Δ, we can query it with (q, π). Thus, in $Q_b(n)$ time, for every subdivision on path π, we find the cell of the cutting that contains q. Then, we use the point location data structure on the conflict lists of the cells to find the original rectangle containing q. This takes an additional $O(\log(f(\Delta)))$ as the size of each conflict is $O(f(\Delta))$. Thus, the query time in this case is

$$Q_b(n) + O(|\pi|\log(f(\Delta))) = Q_b(n) + O(\sqrt{|\pi|}\log n)$$

since we have $|\pi| \leq \frac{1}{2} \cdot \left(\frac{\log n}{\delta_1}\right)^2 = \frac{1}{2} \cdot \left(\frac{\log n}{\log(f(\Delta))}\right)^2$.

Thus the only paths we cannot answer yet are those when $\frac{1}{2} \cdot \left(\frac{\log n}{\delta_1}\right)^2 \leq |\pi| \leq \frac{\log^2 n}{2}$. We bootstrap in this case. First, observe that we can build a data structure $\mathcal{A}'$ on the the original rectangles, where $\mathcal{A}'$ is an instance of the base data structure but this time with parameter Δ set to δ_1^2. This will take $S_b(\delta_1^2, n) = O(nf(\delta_1^2))$ space. Thus, the total space consumption is

$$O(n) + S_b(\delta_1^2, n) = O(n) + O(nf(\log^2(f(\Delta))))$$
$$= O(n) + O(nf(f(\Delta)))$$

where the last inequality follows since $f(\cdot)$ is a monotone increasing function and $\log^2(f(\Delta)) < f(\Delta)$ as $f(\Delta) = \omega(1)$. By construction, the data structure $\mathcal{A}'$ is built to handle exactly paths of this length but it is using too much space. The idea here is that we can repeat the previous technique using "second level" cuttings to obtain a data structure $\mathcal{A}_2$: for a subdivision of size n_i, build a $(f(f(\Delta))/n_i)$-cutting, called the "second level" cutting. By repeating the same idea we used for the first level cuttings, we can spend additional $O(n)$ space to build a data structure $\mathcal{A}_2$ which can answer queries (q, π) as long as $\frac{1}{2} \cdot \left(\frac{\log^2 n}{\Delta}\right) \leq |\pi| \leq \frac{1}{2} \cdot \left(\frac{\log n}{\delta_2}\right)^2$ where $\delta_2 = \log(f(f(\Delta)))$. By repeating this process for $f^*(\Delta)$ steps, we can obtain the claim data structure. ∎

We will begin with the data structure in Lemma IV.4 and use Lemma IV.5 to bootstrap. To facilitate the description, we define two useful functions first. Let $\log^* n$ be the iterated

log function, i.e., the number of times we need to apply log function to n until we reach 2. We define $\log^{*(i)} n$ as follows

$$\log^{*(i)} n = \begin{cases} \log^{*(i)}(\log^{*(i-1)} n) + 1 & n > 1, \\ 0 & n \leq 1. \end{cases}$$

In other words, it is the number of times we need to apply $\log^{*(i-1)}$ function to n until we reach 2. We also define:

$$\tau(n) = \{\min i : \log^{*(i)} n \leq 3\}.$$

In fact, $\log^{*(i)} n$ is the $(i+2)$-th function of the inverse Ackermann hierarchy [19] and $\tau(n) = \alpha(n) - 3$, where $\alpha(n)$ the inverse Ackermann function [19].

Lemma IV.6. *Consider a catalog tree of height h, $\frac{\log n}{2} < h \leq \frac{\log^2 n}{2}$, in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can build a data structure using $O(n\alpha_c(n))$ space, where $c \geq 3$ is any constant and $\alpha_c(n)$ is the c-th function of the inverse Ackermann hierarchy, such that given any query (q, π), where q is a query point and π is a path of length l, $\frac{\log n}{2} < l \leq \frac{\log^2 n}{2}$, all regions containing q along π can be reported in time $O(\sqrt{|\pi|} \log n)$. Furthermore, we can also build a data structure using $O(n)$ space answering queries in time $O(\alpha(n)\sqrt{|\pi|} \log n)$, where $\alpha(n)$ is the inverse Ackermann function.*

Proof: By Lemma IV.4, if we set $h_1 = \frac{\log n}{2}$ and $h_2 = \frac{\log^2 n}{2}$, we obtain a data structure using $O(n \log \log n)$ answering queries in time $O(\sqrt{|\pi|} \log n)$. By picking $\Delta = \log n$, $f = \log n$, we can apply Lemma IV.5 to reduce the space to $O(n \log^*(\log n)) = O(n \log^* n)$ while achieving the same query time. If we again pick $\Delta = \log n$, but $f = \log^* n$, by applying Lemma IV.5 again, the space is further reduced to $O(n \log^{**}(\log n)) = O(n \log^{**} n)$. We continue this process until $\log^{*(i)} n$ is less than three. Note that we will need to pay $O(\sqrt{|\pi|} \log n)$ extra query time each time we apply Lemma IV.5. We will end up with a linear-sized data structure with query time $O(\tau(n)\sqrt{|\pi|} \log n) = O(\alpha(n)\sqrt{|\pi|} \log n)$. On the other hand, if we stop applying Lemma IV.5 after a constant c many rounds, we will end up with a $O(n \log^{*(c)} n) = O(n\alpha_{c+2}(n))$ sized data structure with the original $O(\sqrt{|\pi|} \log n)$ query time. ∎

2) The Lower Bound: We show an almost matching lower bound in this section.

Lemma IV.7. *Assume, given any catalog tree of height h, $\frac{\log n}{2} < h \leq \frac{\log^2 n}{2}$, in which each vertex is associated with a planar subdivision with n being the total complexity of the subdivisions, we can build a data structure that satisfies the following: it uses at most $n2^{\varepsilon \log n/\sqrt{h}}$ space, for a small enough constant ε, and it can answer 2D OFC queries (q, π). Then, its query time must be $\Omega(\sqrt{|\pi|} \log n)$.*

Proof: We first describe a hard input instance for a 3D rectangle stabbing problem and later we show that this can

be embedded as an instance of 2D OFC problem on a a tree of height h. Also, we actually describe a tree of height $h' = \frac{h + (\log n)/4}{2} \leq h$. This is not an issue as we can add dummy vertices to the root to get the height to exactly h.

We begin by describing the set of rectangles. Each rectangle is assigned a "class number" and a "group number". The number of classes is $\frac{\sqrt{h}}{2}$ and the number of groups is $\frac{\log n}{4\sqrt{h}} + \sqrt{h}$. The rectangles with the same class number and group number will be disjoint, isometric and they would tile the unit cube. Rectangles with class $i = 0, \cdots, \frac{\sqrt{h}}{2} - 1$ and group $j = 0, \cdots, \frac{\log n}{4\sqrt{h}} - 1$ will be of shape

$$[\frac{1}{K^j} \times K^j \cdot 2^{ir+j} \cdot V \times \frac{1}{2^{ir+j}}],$$

where V, K are some parameters to be determined later and $r = \frac{\sqrt{\log n}}{4\sqrt{h}}$. Similarly, rectangles with class $i = 0, \cdots, \frac{\sqrt{h}}{2} - 1$ and group $j = \frac{\log n}{4\sqrt{h}}, \cdots, \frac{\log n}{4\sqrt{h}} + \sqrt{h} - 1$ will be of shape

$$[\frac{1}{K^j} \times K^j \cdot 2^{ir+r} \cdot V \times \frac{1}{2^{ir+r}}].$$

The total number of different shapes is $\frac{\sqrt{h}}{2} \cdot (\frac{\log n}{4\sqrt{h}} + \sqrt{h}) = h'$. Note that each rectangle has volume V, so the total number of rectangles we use in all the tilings is n by setting $V = h'/n$. By our construction any query point is contained in $t = h' = |\pi|$ rectangles. Now we analyze the maximal intersection volume of two rectangles. By the same argument as in the proof of Lemma IV.2 the maximal intersection volume can only be achieved by two rectangles when they are in the same class and adjacent groups or in the same group of adjacent classes. For two rectangles R_1 and R_2 in group j_1, j_2 of class i_1, i_2, we have

$$V(R_1 \cap R_2) = \begin{cases} \frac{V}{2K} & i_1 = i_2 \text{ and } j_1 = j_2 + 1 \leq \frac{\log n}{4\sqrt{h}}, \\ \frac{V}{K} & i_1 = i_2 \text{ and } \frac{\log n}{4\sqrt{h}} \leq j_1 = j_2 + 1, \\ \frac{V}{2^r} & i_1 = i_2 + 1 \text{ and } j_1 = j_2. \end{cases}$$

We set $K = 2^r$, then the intersection of any two rectangle is no more than $v = V/2^r$.

We also need to make sure no side length of any rectangle exceeds the side length of the unit cube. The maximum side length can only be obtained when $i = \frac{\sqrt{h}}{2} - 1$ and $j = \frac{\log n}{4\sqrt{h}} + \sqrt{h} - 1$ in the second dimension. We must have

$$K^{\frac{\log n}{4\sqrt{h}} + \sqrt{h} - 1} \cdot 2^{r(\frac{\sqrt{h}}{2} - 1) + \frac{\log n}{4\sqrt{h}}} \cdot V \leq 1$$

Plugging $K = 2^r$ and $V = h'/n$ in, we must have

$$2^{r\frac{\log n}{4\sqrt{h}} + r(\sqrt{h} - 1)} \cdot 2^{r(\frac{\sqrt{h}}{2} - 1) + \frac{\log n}{4\sqrt{h}}} \cdot h' \leq n \qquad (3)$$

Since $\frac{\log n}{2} \leq h \leq \frac{\log^2 n}{2}$ and $r = \frac{\log n}{4\sqrt{h}}$, (3) holds.

Suppose $\gamma h \geq Q(n)$, then the first condition of Theorem II.3 is satisfied and we get the lower bound of

$$S(n) = \Omega(\frac{tv^{-1}}{2^{O(\gamma)}}) = \Omega(\frac{n2^r}{2^{O(\gamma)}}).$$
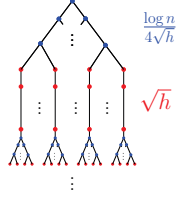
$$\frac{\log n}{4\sqrt{h}}$$

$$\sqrt{h}$$

Figure 2: A difficult tree for fractional cascading

Observe that by setting $\gamma = \frac{\delta \log n}{\sqrt{h}}$ for a sufficiently small $\delta > 0$, the data structure must use $\Omega(n2^{\Omega(\log n/\sqrt{h})})$ space, which contradicts the space usage in our theorem. Therefore, $Q(n) \geq \gamma h = \frac{\delta \log n}{\sqrt{h}} h = \Omega(\sqrt{h} \log n) = \Omega(\sqrt{|\pi|} \log n)$. It remains to show that this set of rectangles can be embedded into an instance of the 2D OFC problem. To do that, we describe the tree T for this embedding. See Figure 2. We hold the convention that the root of T has depth 0. Starting from the root, until depth $\frac{\log n}{4\sqrt{h}}$, every vertex will have two children (blue vertices in Figure 2) then we will have $\sqrt{h}$ vertices with one child (red vertices in Figure 2). Then this pattern continues for $\frac{\sqrt{h}}{2}$ steps. The first set of blue and red vertices correspond to class 1, the next to class 2 and so on. Within each class, the top level corresponds to group 1 and so on. To be specific, vertices at depth $(\frac{\log n}{4\sqrt{h}} + \sqrt{h})i + j$ of the tree have rectangles of class i and group j. Now it can be seen that the rectangles can be assigned to the vertices of T, similar to how it was done in Lemma IV.2. The notable difference is that the depth ((length of the side parallel to the z-axis)) of the rectangles decreases as the group number increases from 0 to $\frac{\log n}{4\sqrt{h}} - 1$ but then it stays the same from $\frac{\log n}{4\sqrt{h}}$ until $\frac{\log n}{4\sqrt{h}} + \sqrt{h} - 1$ but this exactly corresponds to the structure of the tree T. ∎

C. Trees of height $h > \frac{\log^2 n}{2}$

For trees of this height, we have:

Lemma IV.8. *Consider a catalog tree of height $h > \frac{\log^2 n}{2}$ in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can build a data structure using $O(n)$ space such that given any query (q, π), where q is a query point and π is a path of length $|\pi| > \frac{\log^2 n}{2}$, all regions containing q along π can be reported in time $O(|\pi|)$.*

Proof: We combine the classical heavy path decomposition by Sleator and Tarjan [28] and the data structure for catalog paths. We first apply the heavy path decomposition to the tree and then for every heavy path created we build a 2D OFC data structure to answer queries along the path. Clearly, we spend linear space in total. By the property of the heavy path decomposition, we only need to query $O(\log n)$ heavy paths to answer a query, so we obtain $O(\log^2 n + |\pi|) = O(|\pi|)$ query time. ∎

By combining Lemma IV.1, Lemma IV.6, Lemma IV.8, we immediately get the following corollary.

Corollary IV.1. *Consider a catalog tree in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can build a data structure using $O(n\alpha_c(n))$ space, where $c \geq 3$ is any constant and $\alpha_c(n)$ is the c-th function of the inverse Ackermann hierarchy, such that given any query (q, π), where q is a query point and π is a path, all regions containing q along π can be reported in time $O(\log n + |\pi| + \min\{|\pi|\sqrt{\log n}, \sqrt{|\pi|} \log n\})$. Furthermore, we can also build a data structure using $O(n)$ space answering queries in time $O(\log n + |\pi| + \min\{|\pi|\sqrt{\log n}, \alpha(n)\sqrt{|\pi|} \log n\})$, where $\alpha(n)$ is the inverse Ackermann function.*

V. Queries on Catalog Graphs and Subgraph Queries

In this section, we consider general catalog graphs and subtree queries on catalog trees. We show that it is possible to build a data structure of space $O(n)$ such that we can save a $\sqrt{\log n}$ factor from the naïve query time of iterative point locations. We also present a matching lower bound.

We begin by presenting a basic reduction.

Lemma V.1. *Given a catalog graph G of m vertices with graph subdivision complexity n and maximum degree d, we can generate a new catalog graph G' with $\Theta(md)$ vertices with graph subdivision complexity $\Theta(dn)$ and bounded degree $O(d^2)$ such that the following holds: given any connected subgraph $\pi \subseteq G$, in time $O(|\pi|)$, we can find a path π' in G' such that the answer to any query $Q_1 = (q, \pi)$ in G equals the answer to query $Q_2 = (q, \pi')$ in G'.*

Proof: The idea is to turn a subgraph query into a path query by using dummy vertices.

We can obtain G' as follows. For every vertex in G, place $2d$ copies of the vertex in G'. All the copies of a vertex are connected in G'. Furthermore, every copy of a vertex v_i is connected to every copy of vertex v_j iff v_i and v_j are connected in G. The maximum degree of G' is thus $O(d^2)$.

Now consider a subraph query π in G. By definition, π is a connected subgraph of G and w.l.o.g., we can assume π is a tree. We can form a walk W from π by following a DFS ordering of π s.t. W traverses every edge of π at most twice and visits every vertex of π. We observe that we can realize W as a path in G' by utilizing the dummy vertices; as each vertex has $2d$ dummy vertices, every visit to a vertex in G can be replaced by a visit to a distinct dummy vertex. ∎

A. The Upper Bound

Formally, we have the following result.

Lemma V.2. *Consider a degree-bounded catalog graph in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can*

build a data structure using $O(n)$ space such that given any query (q, π), where q is a query point and π is a path, all regions containing q along π can be reported in time $O(\log n + |\pi| \sqrt{\log n})$.

Proof: We build the data structure similar to the proof of Lemma IV.1. The only difference is that the degree of a node is bounded by any constant $d \geq 2$. By the same argument in the proof of Lemma IV.1, any node is stored at most $\Theta(r^{\log d} \log r)$ times and we can obtain a $O(n)$ space bounded data structure achieving $O(\log n + |\pi| \sqrt{\log n})$ query time by creating an intersection sensitive $(r^{2 \log d}/n_i)$-cutting for each planar subdivision A_i and balancing the query time of cutting cells and conflict lists. ∎

By Lemma V.1, we can also obtain the following.

Corollary V.1. *Consider a catalog graph in which each vertex is associated with a planar subdivision. Let n be the total complexity of the subdivisions. We can build a data structure using $O(n)$ space such that given any query (q, π), where q is a query point and π is a connected subgraph, all regions containing q along π can be reported in time $O(\log n + |\pi| \sqrt{\log n})$.*

Specifically, it holds when the catalog graph is a catalog tree and the query subgraph is a subtree.

B. The Lower Bound

In this section, we show that the results in Lemma V.2, Corollary V.1 are tight. Like the proof of path queries for catalog trees, we need a reduction from a rectangle stabbing problem to a 2D OFC subtree query problem on catalog trees. But unlike previous proofs, we use an instance of the rectangle stabbing problem in a much higher dimension. We show the lower bound for subtree queries of a catalog tree. By Lemma V.1, this also gives a lower bound for path queries in general catalog graphs.

Lemma V.3. *Assume, given any catalog tree of height $\sqrt{\log n} \leq h \leq \frac{\log n}{2}$ in which each vertex is associated with a planar subdivision with n being the total complexity of the subdivisions, we can build a data structure that satisfies the following: it uses at most $n 2^{\varepsilon \sqrt{\log n}}$ space, for a small enough constant ε, and it can answer 2D OFC queries (q, π), where q is a query point and π is a subtree containing $b = \sqrt{n}/\log n$ leaves. Then, its query time must be $\Omega(|\pi| \sqrt{\log n})$.*

Proof: We define the following special $(2 + b)$-dimensional rectangle stabbing problem. The input consists of n rectangles in $(2 + b)$ dimensions. According to their shapes, rectangles are divided into $h' = h - r$ sets of size n/h' each where r is a parameter to be determined later. The rectangles in each set are further divided into b groups of size $n/(h'b)$ each. All the rectangles in the same group are pairwise disjoint and they together tile the

$(2 + b)$-dimensional unit cube. We put restrictions on the shapes of the input rectangles to make this problem special. For a rectangle in set i, $i = 0, 1, \cdots, h' - 1$, group j, $j = 0, 1, \cdots, b-1$, except for the first two and the $(2+j)$-th dimensions, its other side lengths are all set to be 1. The side length of the $(2+j)$-th dimension is set to be $1/2^{i+r}$. We put restriction on the side lengths of the first two dimensions in set i as follows: For the first group $j = 0$ in this set, we put no restrictions of the first two dimensions as long as they tile the unit cube and the total number of rectangles used for this group is $n/(h'b)$. For an arbitrary group j, we cut the range of the unit cube in the $(2+j)$-th dimension into 2^{i+r} equal length pieces. This partitions the unit cube into 2^{i+r} parts. Note that each part of the unit cube is also tiled by rectangles since we require the side length of the $(2+j)$-th dimension of the rectangles to be $1/2^{i+r}$. If we project the rectangles in each part of the unit cube into the first two dimensions, we obtain 2^{i+r} axis-aligned planar subdivisions. The planar subdivisions we generated for the first group is used as a blueprint for the shape of rectangles in other groups. More specifically, for the remaining groups in set i, we require that the choices of the first two dimensions to give the same set of 2^{i+r} planar subdivisions as the first group. The problem is as follows: Given a point in $(2 + b)$-dimensions, find all the rectangles containing this query point.

We now describe a reduction from this problem to 2D OFC subtree queries on catalog trees. We consider a complete balanced binary tree of height $h = h' + r$. Note that the number of nodes at layer $i+r$ of the tree is the same as the number of different subdivisions we get by projecting a group in set i to the first two dimensions. Since we require all the groups in the same set yield the same set of subdivisions, we can simply attach the subdivisions to the nodes starting from layer r. For nodes in layer smaller than r, we attach them with empty subdivisions. Now let us analyze a rectangle stabbing query q on the rectangle stabbing problem. Consider rectangles in set i, we need to find the rectangle containing q in each of the b groups. In this special rectangle stabbing problem, to find the rectangle in group j containing q, we can find the rectangle by first using the $(2+j)$-th coordinate of q to find the part of the unit cube where q is in, and then finding the output rectangle by a simple planar point location on the projection of the part using the first two coordinates of q. By our construction this is equivalent to choosing a node in layer $i+r$ of the binary tree and to perform a point location query on the subdivision attached to it. Note that the node in layer $i+r+1$ we choose must be one of the children of the chosen node in layer $i+r$. So if we only focus on one specific group j of all sets, the rectangle stabbing query corresponds to a series of point location queries from the root to a leaf in the binary tree we constructed. Similarly, we obtain b such paths if we consider all groups and they together form a subtree of b leaves. The answer to the point location queries along the subtree gives

the answer to the rectangle stabbing problem.

We describe a hard high dimensional rectangle stabbing problem instance. As before, we create rectangles of different shapes to tile the unit cube. But this time, we will consider a $(2 + b)$-dimensional rectangle stabbing problem. For rectangles in class $i = 0, \cdots, h'/r - 1$, supergroup $j = 0, \cdots, r - 1$, we create the following shapes:

$$
\left.
\begin{array}{l}
[\frac{1}{K^j} \times K^j \cdot 2^{ir+j+r} \cdot V \times \frac{1}{2^{ir+j+r}} \times 1 \times 1 \times \cdots \times 1] \\[2mm]
[\frac{1}{K^j} \times K^j \cdot 2^{ir+j+r} \cdot V \times 1 \times \frac{1}{2^{ir+j+r}} \times 1 \times \cdots \times 1] \\[2mm]
\vdots \\[2mm]
[\frac{1}{K^j} \times K^j \cdot 2^{ir+j+r} \cdot V \times 1 \times 1 \times \cdots \times 1 \times \frac{1}{2^{ir+j+r}}]
\end{array}
\right\} b \text{ shapes}
$$

where K, V are parameters to be determined later. Note that all the rectangles are in $(2 + b)$ dimensions.

We use each of the shape to tile a unit cube. Since the volume of any rectangle is V, we need $1/V$ rectangles of the same shape to tile the cube. We call it a group. Note that the rectangles in the same group are pairwise disjoint. We generated $h'b/V$ rectangles in total. By setting $V = h'b/n$, the total number of rectangles is n. Note that any point in the unit cube is contained in exactly $t = h'b$ rectangles.

Now we analyze the volume of the intersection between any two $(2 + b)$-dimensional rectangles. Note that if two rectangles have the same side lengths for $b - 1$ out of the last b dimensions, then it is the case we have analyzed in the proof of Lemma IV.2, and the volume of the intersection of any two rectangles is bounded by V/K if we set $K = 2^r$. Now we analyze the other case. By our construction, two rectangles can only have at most two different side lengths in the last b dimensions. We consider two rectangles in class i_1 supergroup j_1, and class i_2 supergroup j_2 respectively. Without loss of generality, we assume $j_1 \geq j_2$. The case for $j_1 \leq j_2$ is symmetric. Then there are two possible expressions for the intersection volume depending on the values of i_1 and i_2. The first one is

$$
\frac{1}{K^{j_1}} \times K^{j_1} \cdot 2^{i_1 r + j_1 + r} \cdot V \times \frac{1}{2^{i_1 r + j_1 + r}} \times \frac{1}{2^{i_2 r + j_2 + r}}
$$
$$
= \frac{V}{2^{i_2 r + j_2 + r}} \leq \frac{V}{K}.
$$

The second possible expression is

$$
\frac{1}{K^{j_1}} \times K^{j_2} \cdot 2^{i_2 r + j_2 + r} \cdot V \times \frac{1}{2^{i_1 r + j_1 + r}} \times \frac{1}{2^{i_2 r + j_2 + r}}
$$
$$
= \frac{V}{K^{j_1 - j_2}} \times \frac{1}{2^{i_1 r + j_1 + r}} \leq \frac{V}{K}.
$$

The last inequality holds because $j_1 \geq j_2$.

To make this construction well-defined, no side length of the rectangles can exceed 1. The largest side length can only

be obtained in the second dimension when $i = h'/r - 1$ and $j = r - 1$. We must have

$$
K^{r-1} 2^{h'+r-1} V \leq 1.
$$

Plugging in $V = h'b/n$ and $K = 2^r$, then we must have

$$
2^{r^2 - r} 2^{h' + r - 1} h'b < n \tag{4}
$$

Since by our assumptions $h' \leq \frac{\log n}{2}$, $b \leq \frac{\sqrt{n}}{\log n}$, it follows that by setting $r = \frac{\sqrt{\log n}}{4}$, the inequality (4) holds.

If $\gamma h'b \geq Q(n)$ holds, then the first condition of Theorem II.3 is satisfied and we obtain the lower bound of

$$
S(n) = \Omega(\frac{tv^{-1}}{2^{O(\gamma)}}) = \Omega(\frac{n2^r}{2^{O(\gamma)}}).
$$

If we set $\gamma = \delta\sqrt{\log n}$ for a sufficiently small $\delta > 0$, the data structure must use $\Omega(n2^{\Omega(\sqrt{\log n})})$ space, which contradicts the space usage stated in our lemma. Note that $|\pi| = (h' + r)b = \Theta(h'b)$. Then $Q(n) \geq \gamma h'b = \Omega(|\pi|\sqrt{\log n})$. ∎

Combining Lemma V.1 and Lemma V.3, we immediately have the following corollary:

Corollary V.2. *Assume, given any degree-bounded catalog graph in which each vertex is associated with a planar subdivision with n being the total complexity of the subdivisions, we can build a data structure that satisfies the following: it uses at most $n2^{\varepsilon\sqrt{\log n}}$ space, for a small enough constant ε, and it can answer 2D OFC queries (q, π), where q is a query point and π is a path. Then, its query time must be $\Omega(|\pi|\sqrt{\log n})$.*

VI. Open Problems

For the linear space data structure we obtained for general path queries of trees Corollary IV.1, there is a tiny inverse Ackermann gap between the query time we obtain and the lower bound. It is an interesting problem whether we can get rid of that term or improve the lower bound.

The problem we consider is very general in the sense that the only restriction we place on the input instance is that the graph subdivision complexity is n. Some special cases admit better solutions. For example, if we require the subdivision complexity of each vertex of the graph to be asymptotically the same, we can obtain an $O(n)$ space and $O(\log n + |\pi| \log\log n)$ query time data structure [2] for path queries on catalog trees of height $h \leq \frac{\log n}{2}$, which is better compared to the linear space $O(\log n + |\pi|\sqrt{\log n})$ query time data structure we obtained in Lemma IV.1.

Higher dimensional generalization of our results is another direction. In 2D, we can transform an axis-aligned planar subdivision to a subdivision consisting of only rectangles by increasing the subdivision complexity by only a constant

---

[2]This is done by creating an intersection sensitive $(\log n/n_i)$-cutting C_i for each subdivision A_i in the tree and then storing all cutting cells on each path using the data structure in Theorem III.1 and building point location data structures on the conflict list of each cell.

factor; however it is not the case for 3D. On the other hand, for 3D point locations on orthogonal subdivisions, we have Rahul's $O(\log^{3/2} n)$ query time and linear space data structure [27] in the standard pointer machine model. Recently, the query time is improved to $O(\log n)$ by Chan et al. [11], but they use a stronger arithmetic pointer machine model. Given that the higher dimensional counterparts of the tools we use for 2D are suboptimal, it is a challenging problem to see how the results will be in higher dimensions.

Other open problems include considering the dynamization of our results and other computational models.

REFERENCES

[1] P. Afshani. Improved pointer machine and I/O lower bounds for simplex range reporting and related problems. In *Proceedings of the twenty-eighth annual Symposium on Computational Geometry*, pages 339–346, 2012.

[2] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pages 240–246, 2010.

[3] P. Afshani, L. Arge, and K. G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of the twenty-eighth annual Symposium on Computational Geometry*, pages 323–332, 2012.

[4] P. Afshani, T. M. Chan, and K. Tsakalidis. Deterministic rectangle enclosure and offline dominance reporting on the ram. In *International Colloquium on Automata, Languages, and Programming*, pages 77–88. Springer, 2014.

[5] P. Afshani, C. Sheng, Y. Tao, and B. T. Wilkinson. Concurrent range reporting in two-dimensional space. In *Proceedings of the twenty-fifth annual ACM-SIAM Symposium on Discrete algorithms*, pages 983–994. SIAM, 2014.

[6] P. K. Agarwal. Partitioning arrangements of lines II: Applications. *Discrete & Computational Geometry*, 5(6):533–573, 1990.

[7] P. K. Agarwal. *Geometric partitioning and its applications*. Duke University, 1991.

[8] J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244 – 251, 1979.

[9] M. d. Berg and O. Schwarzkopf. Cuttings and applications. *International Journal of Computational Geometry & Applications*, 5(04):343–355, 1995.

[10] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the twenty-seventh annual symposium on Computational geometry*, pages 1–10, 2011.

[11] T. M. Chan, Y. Nekrich, S. Rahul, and K. Tsakalidis. Orthogonal point location and rectangle stabbing queries in 3-d. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, page 31. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2018.

[12] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

[13] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993.

[14] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[15] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[16] B. Chazelle and L. J. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

[17] B. Chazelle and L. J. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1(1-4):163–191, 1986.

[18] B. Chazelle and D. Liu. Lower bounds for intersection searching and fractional cascading in higher dimension. *Journal of Computer and System Sciences*, 68(2):269–284, 2004.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[20] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3 edition, 2008.

[21] M. De Berg, M. van Kreveld, and J. Snoeyink. Two-and three-dimensional point location in rectangular subdivisions. In *Scandinavian Workshop on Algorithm Theory*, pages 352–363. Springer, 1992.

[22] P. F. Dietz and R. Raman. Persistence, amortization and randomization. In *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, page 78–88, 1991.

[23] Y. Giyora and H. Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Transactions on Algorithms*, 5(3), 2009.

[24] J. Matoušek. Cutting hyperplane arrangements. *Discrete & Computational Geometry*, 6(3):385–406, 1991.

[25] J. Matoušek. Approximations and optimal geometric divide-and-conquer. *Journal of Computer and System Sciences*, 50(2):203–208, 1995.

[26] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990.

[27] S. Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in $\mathbb{R}^3$. In *Proceedings of the twenty-sixth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 200–211. SIAM, 2014.

[28] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.